SDN Controller Load Balancing Based on Reinforcement Learning

Zhuo Li[1,2], Xu Zhou[1], Junruo Gao[1,2], Yifang Qin[1]
[1]Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China
[2]University of Chinese Academy of Sciences, Beijing 100049, China
{lizhuo, zhouxu, gaojunruo, qinyifang}@cnic.cn

***Abstract*—Aiming at the local overload of multi-controller deployment in software-defined networks, a load balancing mechanism of SDN controller based on reinforcement learning is designed. The initial paired migrate-out domain and migrate-in domain are obtained by calculating the load ratio deviation between the controllers, a preliminary migration triplet, contains migration domain mentioned above and a group of switches which are subordinated to the migrate-out domain, makes the migration efficiency reach the local optimum. Under the constraint of the best efficiency of migration in the whole and without migration conflict, selecting multiple sets of triples based on reinforcement learning, as the final migration of this round to attain the global optimal controller load balancing with minimum cost. The experimental results illustrate that the mechanism can make full use of the controllers' resources, quickly balance the load between controllers, reduce unnecessary migration overhead and get a faster response rate of the packet-in request.**

***Index Terms*—load balancing, software-defined networking(SDN), switch migration, reinforcement learning**

I. Introduction

Software-defined networking (SDN) separates the control plane and data plane of network devices [1], whose refined centralized control and network programmability bring unique advantages compared with traditional networks. Along with the explosive growth of network traffic and the emergence of network applications of big data, strictly centralized control planes can't meet the requirements of the existing network, the semi-centralized or logically centralized control plane and the fully distributed control plane become the development trend of future enterprise network and carrier network architecture [2], [3].

The distributed deployment based on multiple controllers improves the scalability of the control plane and avoids single point failures in centralized deployment [4]. However, the distributed deployment method is likely to lead some controllers in the control plane to be overloaded while the other controllers are relatively idle [5], [6]. In order to make full use of the forwarding capability of the controller on the control plane, a SDN controller load balancing mechanism is proposed by migrating switch between controllers [7]. The selection algorithm selects a series of switch migration triples (the migrate-out domain, the migrate-in domain, and the migrating switch), then establishes an optimal migration model based on reinforcement learning in the switch migration decision phase [8]. From local optimization to total optimization, the final switch migration action can achieve the purpose of global load balancing.

II. Related Works

A. Related Research Review

Research on SDN controller load balancing has been around for a long time, but the traditional switch migration mechanism can't balance the relationship between load ratio deviation change and migration overhead between controllers. The survey found that the majority of the domestic and foreign literatures are based on switch migration when designing the load balancing mechanism. However, there are several problems in the migration of switches, staid migration selection objects, inefficient migration and migration conflicts.

In order to solve the scalability problem of the SDN control plane better, the researchers have proposed a mechanism of the dynamic migration based on multiple controllers [9], which avoids a single point of failure or excessive controller load when the controller is faulty or overloaded. A dynamic mapping can improve efficiency in managing traffic load variations [10], [11], while the efficiency is increased, the probability of migration conflict and new overload is also great improved. A load balancing mechanism based on a strategy of load informing for multiple distributed controllers, load notifications bring a lot of communication overheads to the entire system [12]. Starting from the latest status of the research at home and abroad, this paper attempts to combine the reduction of the overall load deviation coefficient with the migration efficiency, which dynamically migrates switches in a distributed deployment control platform to avoid single point failure caused by controller overload, the SDN controller load balancing mechanism based on reinforcement learning achieves the following three objectives: (i). better load balancing, (ii). without migration conflicts, (iii). without new overload.

B. Load Balancing Mechanism

When a controller generates an overload, it is generally because a large number of switches cannot find a corresponding flow entry for the newly arrived packet [13], thereby send a packet-in message to the default controller. This paper proposes a load balancing mechanism for the control plane. Each controller of the control plane shares the same network view and achieves load balancing through dynamic migration

of the switch. The dynamic load balancing diagram is shown in Fig. 1, when a controller in the control plane (Controller C_1) has an overload trend, it will enter the switch migration phase, and part of the switch of the overload controller (Switch S_5 and Switch S_9) are migrating to the relatively idle controller (Controller C_2 and Controller C_3).

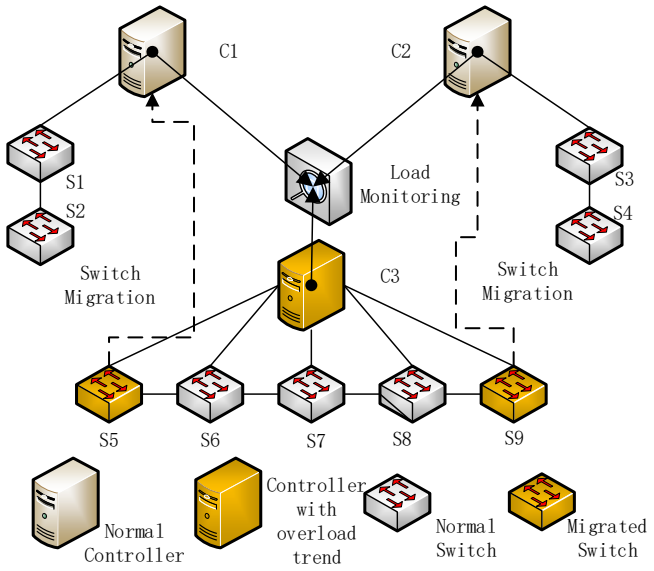

Fig. 1. Controller load diagram.

As the existing network environment is very complex, the real-time load ratio of the controller will change as the complex network environment changes [6]. The load balancing mechanism adopted by the SDN control platform shouldn't simply change the load value of each controller.

By obtaining the packet-in message processing situation of the controller [4], the receiving rate of the controller's packet-in message, combined with the load capacity of each controller, the real-time load ratio of the controller can obtain. When the real-time load ratio of one or several controllers is greater than the rate of the mean load of all controllers so that the equilibrium state is broken, thereby going the switch migration phase. Under the constraint of without migration conflict and load capacity [13], part of the switches of the overload controller is migrating to the relatively idle controller. Fig. 2 is the load balancing mechanism designed in this paper.

In the following chapters, the third chapter introduces the design of switch migration, including the judgment of SDN controller load balancing, switch migration efficiency and the selection process of the switch migration domain and migrating switch. The fourth chapter introduces the optimal migration model and the switch migration decision process based on reinforcement learning. The fifth chapter designs a targeted simulation experiment facing the load balancing mechanism proposed in this paper, gives and analyzes the results in comparison experimental. The sixth chapter gives a brief summary of the full text.

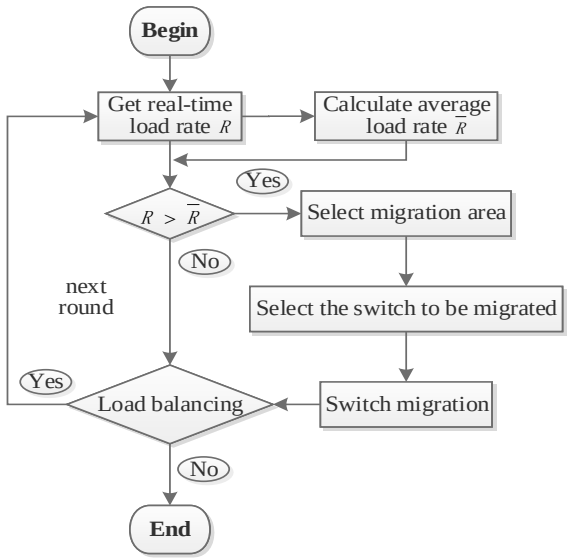

Fig. 2. Load balancing mechanism.

III. Switch Migration Design

Through the switch migration, the load ratio of the controller with overloading trend can be reduced [11], [12], thereby reducing the probability of occurrence of the overloaded controller and maximizing the utilization of the resources in current controller to meet the service request from the switch. Controller's load, controller's load ratio, discrete coefficient and migration efficiency are defined in a series of switch migration models during the switch migration phase. The selection algorithms of migration domain and migrating switch are designed to select the migration triples, which can be used in the next step of the reinforcement learning process. Some of the mathematical symbols used in this section and their meanings are shown in Table I:

TABLE I
The Meaning of the Symbol

Symbol	Meaning
L_{C_i}	the load of the controller C_i
C_{C_i}	the load capacity of the controller C_i
R_{C_i}	the load ratio of the controller C_i
L_{S_k}	the packet-in message sending rate of the switch S_k
$\overline{R}$	mean load ratio of all controllers
D	load ratio deviation coefficient between two controllers
E	migrate switch efficiency between controllers
H_{S_k}	number of hops between the switch and the controller

A. Load Balancing

a) Controller's Load: The load of the SDN controller mainly comes from the following aspects [6], [10], [12]: (i). The switch sends packet-in message to the controller while it can't find the corresponding forwarding flow entry. (ii).

Controllers communicate with each other in the control plane. (iii). The controller installs flow entries for the switch. (iv). All controllers maintain and share the same network view of the entire control platform. The number of packet-in messages received by the controller is the primary source of controller load, this article simplifies the controller load to the sum of the controller's packet-in message rates.

$$L_{C_i} = \sum_{k=1}^{l} L_{S_k} \tag{1}$$

b) Controller's Load Ratio: Different controllers have different CPU performance, number of processors, memory size, etc., which results in different load capacities. The controllers' load ratio R_{C_i} is the ratio of the controller's real-time load L_{C_i} to the controller's load capacity C_{C_i}:

$$R_{C_i} = L_{C_i} / C_{C_i} \tag{2}$$

Obviously, the mean load ratio between two of the controllers:

$$\bar{R}_{(C_i,\ C_j)} = \left(R_{C_i} + R_{C_j}\right)/2 \tag{3}$$

Furthermore, the mean load ratio of all controllers:

$$\bar{R} = \sum_{i=1}^{n} R_{C_i} / n \tag{4}$$

c) Load Balancing Between Controllers: The discrete coefficient is used to describe the degree of discrete of the controller's load ratio and the mean load ratio between controllers. Here, the two controller discrete coefficients can be defined as:

$$D_{(C_i,\ C_j)} = \left(\sqrt{\sum_{k=i,\ j} \left(R_{C_k} - \bar{R}_{C_i,\ C_j} \right)^2 / 2} \right) / \bar{R}_{C_i,\ C_j} \tag{5}$$

Similarly, it is extended to multiple controllers with discrete coefficients:

$$D = \sqrt{\left(\sum_{i=1}^{n} \left(R_{C_i} - \bar{R} \right)^2 / n \right)} / \bar{R} \tag{6}$$

The discrete coefficient of all controllers is defined as the degree of the load balance of the entire control plane, which will become one of the indicators for evaluating the performance of the selection algorithm. The smaller the value is, the better the load balancing of the control plane.

B. Switch Migration Efficiency

If the switch is migrating between controllers, there will be migration overhead. Because the hop count H_{S_k} between the switch and the controller is different, the load L_{S_k} of the switch to the controller is different, the migration efficiency is expressed as follows.

$$E_{(C_i,\ S_k)} = L_{S_k} / H_{S_k} \tag{7}$$

The mean efficiency of migration in the next set of switches from the out-migration controller C_i is as follows:

$$\begin{cases} \bar{E}_{(C_i,S)} = \sum_{k=1}^{l} L_{S_k} / H_{S_k} \\ S = \{S_1, S_2, \cdots\cdots S_l\}, \quad \forall S_k \in C_i \end{cases} \tag{8}$$

Where $S_k \in C_i$ is represented here as the controller of the switch S_k is C_i.

C. Migration Tuples Selection

a) Migration domain selection: In a real-time changing network environment, the load of each controller also changes fast. The traditional method tends to strip the selection process of the out-migration domain and the in-migration domain, which introduces the probability of controller overload to some extents, while it is also easy to bring migration conflicts and migration jitter in the control plane. Next, a new selection algorithm will be proposed and which will add anti-collision mechanism in the judgment part to avoid the two consequences of the commonly bad migration. The switch migration domain selection algorithm is as follows:

Algorithm 1 Selection Algorithm of Migration Domain

Input: Controllers' load ratio $R = \{R_{C_1}, R_{C_2}, \cdots\cdots R_{C_n}\}$,
Mean load ratio of all controllers $\bar{R}$,
Discrete coefficient between the controllers D
Output: Out-migration domain O, in-migration domain I, migration queue Q
Initialization: Set O, I and Q to empty sets

```
for R_{C_i}, R_{C_j} ∈ R = {R_{C_1}, R_{C_2}, ····· R_{C_n}} do
    Calculate the discrete coefficients D_{C_i, C_j}
    if D_{C_i,C_j} > D && R_{C_i} > R̄ && R_{C_i} > R_{C_j} then
        Add C_i to the out-migration domain O
        Add C_j to the in-migration domain I
    end if
    if D_{C_i,C_j} < D && R_{C_i} < R̄ && R_{C_i} < R_{C_j} then
        Add C_j to the out-migration domain O
        Add C_i to the in-migration domain I
    end if
    Add (C_i, C_j) to the migration queue Q
end for
```

b) Migration switch selection: For each migration queue Q that will be migrating, the migration switch selected by the algorithm is defined as the fastest reduction of the factor of the load ratio under the best efficiency E_{C_i,S_k} in migration. The switches under the out-migration controller are sorted according to the migration efficiency, the higher the efficiency of the switch migration, the higher the possibility of being selected. The constraints for selecting a switch for migration

are as follows:

$$\begin{cases} R'_{C_x} = R_{C_x} \mp \sum_{k=1}^{l} L'_{S'_k} \times (C_i/C_j) \\ D_{(C_i,\ C_j)} = \left(\sqrt{\sum_{k=i,\ j} \left(R'_{C_x} - \bar{R}_{C_i,\ C_j} \right)^2 /2} \right) / \bar{R}_{C_i,\ C_j} \\ argmin\ D_{(C_i,\ C_j)} \\ \forall E_{(C_i,S'_k)} \geq \bar{E}_{(C_i,S)} \\ \forall E_{(C_i,S'_k)} \geq E_{(C_i,S'_{k+1})} \\ x = i\ or\ j \end{cases} \tag{9}$$

A switch combination managed by the out-migration controller, whose migration efficiency of each switch has greater than the average efficiency in migration and arranged by size, which is finding out by the selection algorithm. It is the optimal migration switch combination when it maximizes the reduction of the discrete coefficients between controllers. The specific algorithm of selecting migrating switch is shown as follows:

Algorithm 2 Selection Algorithm of Migrating Switch

Input: Migration queue Q
Output: Switch collection S', migration tuples T
Initialization: Set T to an empty set, $n = 0$

for $q = \{C_i, C_j\} \in Q$ **do**
  for $S_k \in C_i$ **do**
    Calculate E_{S_k}
    $E_{C_i,S} \leftarrow E_{C_i,S} + E_{C_i,S_k},\ L \leftarrow E_{C_i,S_k}$
  end for
  sort L // Sort the migration efficiency in L
  $\bar{E}_{C_i,S} \leftarrow E_{C_i,S}/n,\ k = 1$
  while $E_{C_i,S} \in L$ && $S_k \in S$ **do**
    Calculate R'_{C_i}, R'_{C_j} and D'_{C_i,C_j}
    if $E_{C_i,S_k} \geq \bar{E}_{C_i,S}$ && $D'_{C_i,C_j} < temp$ **then**
      $k++,\ temp \leftarrow D'_{C_i,C_j}$, add S_k to S'
    else
      break // Jump out of the loop
    end if
  end while
  $T \leftarrow S'$ // Add collection to the tuples
end for

IV. Switch Migration Decision

A. Optimal Switch Migration Model

With constantly searching strategy, an optimal migration model with the highest efficiency of switch migration and without migration conflicts is composed of the migration queue obtained in the previous chapter. When the migration efficiency can't be optimized by using the greedy strategy only to make decisions [8], [14], exploring non-greedy situations at the case of probability in each decision and combining ε-greedy and decision-making to select migration triples, so as to obtain global optimal switch migration decision based on reinforcement learning [15], [16]. Fig. 3 is the optimal model of switch migration.

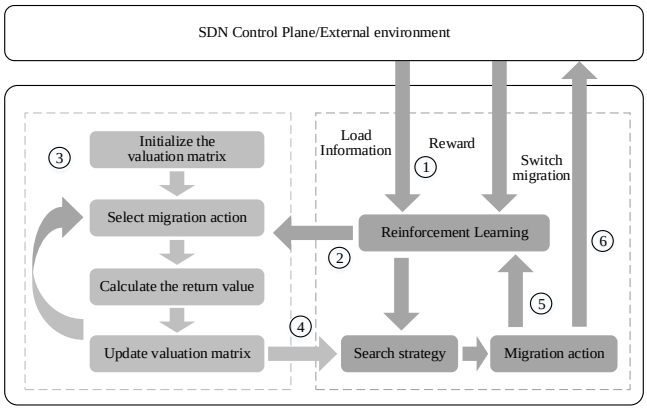

Fig. 3. Optimal migration model based on reinforcement learning.

In this case, the decision-making process in reinforcement learning is defined as taking action based on the current state to obtain the return value as the core driving force for optimizing the migration model [17], [18]. Specifically, the migration domain is the state space to which the current state belongs, and the switch migration is the action taken. The return value of each migration triples is as follows:

$$R_{i,j} = \left(D'_{i,j} - D_{i,j}\right) / \sum_{t=1}^{k} Cost\left(S_t\right) \tag{10}$$

In the expression of the return value [16], the numerator is the degree of change of the load balancing after migration, and the denominator is the sum of the migration costs of multiple groups of migrations in the migration decision. The larger the load balancing rate of reduction value is, the less the overhead and the greater the return value.

The value of $Q\left(s,a\right)$ is iteratively updated by selecting the largest item in each row of the ε greedy strategy, and Q is an evaluation matrix. The iteratively updated method of the valuation matrix is shown as follows:

$$\begin{aligned} Q(s,a)_{t+1} =& Q(s,a)_t + \alpha(\gamma + \gamma \times max_a Q(s',a')_t \\ & - Q(s,a)_t) \end{aligned} \tag{11}$$

The sum of the current return value γ and the next largest maximum value $max_a Q\left(s^{'}, a'\right)_t$ in the memory is used as the expected value of the valuation, and the incremental iterative learning is performed by using the difference between the expected expectation and the true estimate, so as to obtain the value of $Q\left(s,a\right)_t$ in the $(t+1)^{th}$ round.

It is a number greater than 0 and less than 1, the lager the value of the learning ratio R is, the smaller the result in previous training is, while the value of 0 means that the result in previous training is not considered. In addition, the greater the value of the discount factor γ is, the greater consideration of long-term benefits, the smaller the size of the immediate consideration. In order to select the optimal set

of migration triplet $\{C_i, C_j, S_k\}$, the matrix R with reward values iteratively updates the valuation matrix until it is stable, and the group with the largest value in Q is the final switch migration set, which can guarantee the optimal migration efficiency and avoid migration conflicts.

B. Migration Decision Based on Reinforcement Learning

The core principle of reinforcement learning is applied to the SDN switch migration decision process, and an optimal migration model based on reinforcement learning is established, where the migration decision is represented by the migration action derived from reinforcement learning. Specifically, the process in migration decision based on reinforcement learning is as follows [16], [18]:

1) Initialization phase: Initialize the set A in migration action, the evaluation matrix Q and the reward matrix R, where all values in the Q are initialized to 1.

2) Reinforcement learning phase: For each potential out-migration domain, it is necessary to determine in advance whether the in-migration domain in the migration triple has been added to the migration target in migration action.

a) : Difference in load rate between two controllers in the migration matrix and the properties of the switches that are migrating in one migration loop, which will cause different real-time reward value R. In the next step, the migration matrix is searched to calculate the return value of the migration triplet, and the in-migration domain C_{in} with the largest value is selected as the migration target of the out-migration domain C_{out} in this phase.

b) : The probability of selecting the largest estimate in round t is ε, while the probability of exploring other valuations for its iterative operation is 1-ε, the migration triplet corresponding to the maximum estimate selected by probability ε during this period will be used in the next step.

c) : After the migration triple selected in the t-th iteration, the maximum valuation $Q\left(C'_{out}, C'_{in}\right)_t$ of the next migration triple $\{C'_i, C'_j, S'_k\}$ is selected by the maximum greedy strategy, and the expected value $r+r\times Q\left(C'_{out}, C'_{in}\right)_t$ of the maximum estimate $Q\left(C_{out}, C_{in}\right)_t$ is calculated by the discount factor γ. The iterative update formula is as follows:

$$Q(C_{out}, C_{in})_{t+1} \leftarrow Q(C_{out}, C_{in})_t + \alpha(\gamma + \gamma \times Q(C_{out}, C_{in})_t - Q(C_{out}, C_{in})_t) \tag{12}$$

The evaluation matrix Q is calculated in the current state, if it is larger than the original value, the global migration optimal constraint is satisfied, otherwise, the other migration triplet will be selected;

d) : Add the migration triplet $\{C_i, C_j, S_k\}$ to the migration set A, and simultaneously update the migration state of the candidate migration matrix, and remove the migration set;

e) : Returning to loop a), the migration action selection process based on reinforcement learning in this phase, which loops in turn until the evaluation matrix Q in loop c) is stable and no longer changes.

3) Ending phase: Through the reinforcement learning phase, the migration set A is selected to perform the final switch migration.

V. Simulation Analysis

A. Simulation Experiment Environment

In the simulation experiment environment of this paper, the performance of load balancing mechanism is analyzed by testing the multi-controller cluster. The controller in control plane is the open source controller RYU [19], which is implemented by the Python language and supports all the mainstream Openflow protocols [20]. It can quickly develop the control that satisfies the load balancing mechanism this article. Mininet [21] build the network topology of the entire simulation experiment environment, and the experimental platform is a virtual machine, whose type is Ubuntu 16.04. Four RYU controllers C1, C2, C3, and C4 are deployed in the controller cluster, and 12 OpenvSwitch [22] are connected to four controllers. Each switch can connect one or more hosts. In the experiment, the host will use the iPerf [23] to simulate the data packet with different rates, and the simulation experiment topology is as follows in Fig. 4.

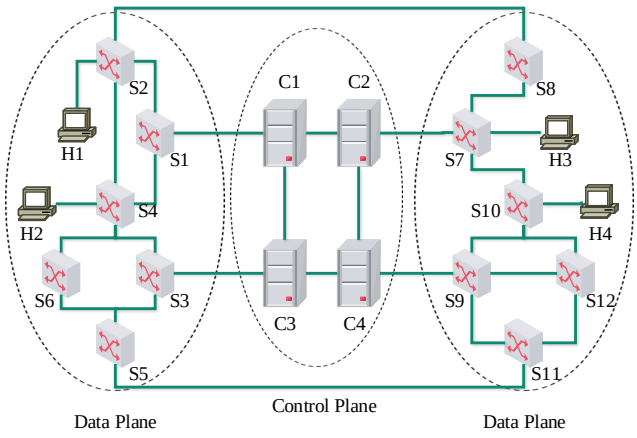

Fig. 4. Simulation experiment topology.

In the above environment, the load measurement module on the Mininet platform can measure the number of packet-in events sent to the controller per unit time, that is, the ratio of packet-in event sending, and the number of hops from the switch to the controller is defined in the topology file of the Mininet platform.

B. Simulation Experiment Analysis

In order to fully analyze the performance of the switch migration mechanism designed in this paper, a comparative experiment was specially added to the simulation experiment. According to the load balancing mechanism based on reinforcement learning (RL-LBM) proposed in this paper, two typical mechanisms of load balancing are selected as the comparison objects. One typical mechanism (DC-LBM) is to

preferentially migrate the switch with the highest transmission rate of the packet-in event under the overload controller based on the deviation coefficient of the controllers' load rate. Another representative mechanism (MMO-LBM) is the load balancing of switch migration based on minimum migration overhead, that the overload controller migrates a part of the switch to the neighbor controller with the fewest hops to achieves.

Compared with the above two comparison mechanisms, the experimental results show that the load balancing mechanism designed in this paper can effectively migrate the switch under the controller with trend overload to the relatively idle controller, which improves the utilization ratio of controllers' resource, reduces the response time of the packet-in event of the switch and improves the load balancing effect in the whole network environment. At the same time, the migration overhead caused by switch migration does not bring extra burden to the whole system, which verifies the superiority of the mechanism in this paper.

a) Load balancing Ratio: When different mechanisms for load balancing are compared together, the ratio in load balancing is a key indicator to measure them, and each of them aims to reduce the rate in load balancing of the control plane. With the rate in different transmission of the iPref, the rate of load balancing of the three mechanisms in the same period is as follows in Fig. 5.

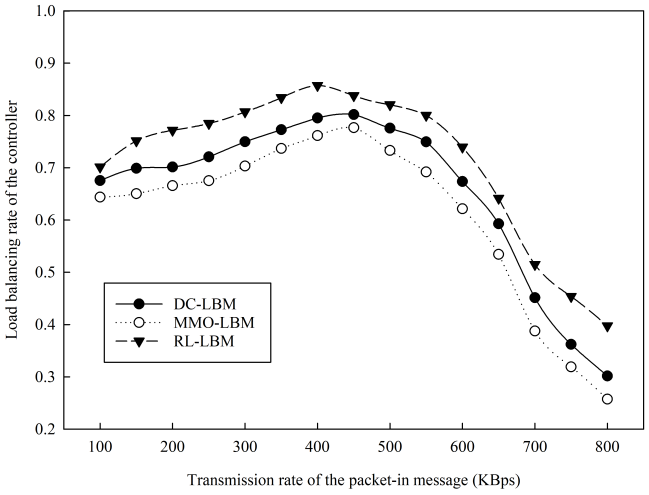

Fig. 5. Load balancing rate in control plane.

After using the mechanism of switch migration, the rate in load balancing of each controller is improved. As the packet-in message transmission rate continues to increase, whose value is reduced due to the constraints of the controller's capability.

b) Packet-in Processing Delay: When the switch migration is initiated, the controller in out-migration domain sends a flow-mod message to the controller in in-migration domain, and the management control of the switch will change when it arrives. Due to the change of the transmission link and the installation time of the new flow table, the delay in processing the packet-in message will inevitably increase. The specific performance of the various mechanisms is as follows:

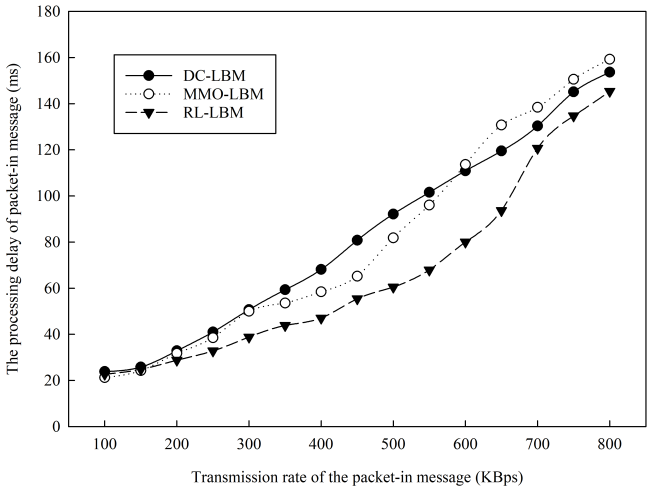

Fig. 6. Processing delay of the packet-in message.

According to the experiment, the processing delay of the packet-in increases, as the transmission rate of the packet-in message increases. It can be seen from Fig. 6 that the switch-controller message response delay of the RL-LBM mechanism is reduced by 17.8% on average compared with the MMO-LBM.

c) Packet-in Processing Delay: Switch migration inevitably brings migration overhead to the entire system, whose size is directly related to the number of migrations and the number of controller-to-switch hops, and continuous switch migration increases migration overhead with the operation of the load balancing mechanism. Fig. 7 shows the migration overhead of various mechanisms:

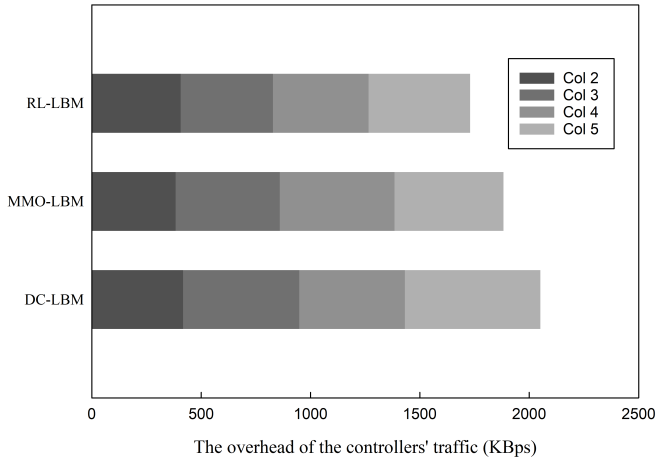

Fig. 7. Switch migration overhead.

Under the same network conditions, switch action taken by different load balancing mechanisms will directly affect the overhead of the switch migration, which includes the controller's traffic overhead and controller synchronization delay. The experimental results in Fig. 7 show that the RL-LBM mechanism has obvious advantages over the other two, whether in a single controller or the entire system in terms of migration overhead.

VI. Conclusion

This paper designs a selection algorithm in the phase of switch migration, which is used to select the out-migration domain, the in-migration domain and the switch to be migrating, as combined into one switch migration triplet. By using reinforcement learning, the load balancing mechanism chooses a set of switch migration queues that can achieve global optimization from local optimization. Simulation experimental results illustrate that the mechanism can make full use of the controllers' resources, quickly balance the load between controllers, reduce unnecessary migration overhead and get a faster response rate of the packet-in request.

Acknowledgment

This work was supported in part by the National Science and Technology Major Project of the Ministry of Science and Technology of China under Grant 2017ZX03001015, Grant 2018ZX03001015 and Grant 2018ZX03001021.

References

[1] T. D. Nadeau and K Gray, "SDN: Software Defined Networks," California: O'Reilly Media, 2013.
[2] T. Huang, J. Liu, L. Wei and J. Zhang, F. Yang and Y. Liu, "SDN Core Principles and Application Practice," Beijing: the People's Posts and Telecommunications Press, 2016.
[3] Q. Zuo, M. Chen, G. Zhao, C. Xing, G. Zhang and P. Jiang, "Research on OpenFlow-Based SDN Technologies," Journal of Software, vol. 05, pp. 1078-1097, 2013.
[4] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman and R. R. Kompella, "ElastiCon: an elastic distributed SDN controller," 2014 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), Marina del Rey, CA, 2014, pp. 17-27.
[5] P. Song, Y. Liu, T. Liu, D. Qian, "Flow Stealer: lightweight load balancing by stealing flows in distributed SDN controllers," SCIENCE CHINA Information Sciences, 2017, 60(3): 032202.
[6] Y. Wang, J. Yu, K. Pei and X. Qiu, "A Load Informing Based Load Balancing Mechanism for Multiple Controllers in SDN," Journal of Electronics and Information Technology, vol. 11, pp. 2733-2740, 2017.
[7] M. Cello, Y. Xu, A. Walid, G. Wilfong, H. J. Chao and M. Marchese, "BalCon: A Distributed Elastic SDN Control via Efficient Switch Migration," 2017 IEEE International Conference on Cloud Engineering (IC2E), Vancouver, BC, 2017, pp. 40-50.
[8] C. Wang, B. Hu, S. Chen, D. Li and B. Liu, "A Switch Migration-Based Decision-Making Scheme for Balancing Load in SDN," in IEEE Access, vol. 5, pp. 4537-4544, March 2017.
[9] G. Cheng, H. Chen H. Hu and J. Lan, "Dynamic switch migration towards a scalable SDN control plane," International Journal of Communication Systems, vol. 29, no. 9, pp. 1482-1499, February 2016.
[10] W. Li, S. Shen and Z. Wu, "A Switch Migration Mechanism Based on Multiple SDN-controllers," Computer Technology and Development, vol. 01, pp. 89-94, 2018.
[11] G. Cheng, H. Chen, H. Hu and Z. Wang, "Toward a scalable SDN control mechanism via switch migration," China Communications, vol. 14, no. 1, pp. 111-123, January 2017.
[12] X. Ye, G. Cheng and X. Luo, "Maximizing SDN Control Resource Utilization via Switch Migration," Computer Networks, vol. 126, no. 24, pp. 69-80, October 2017.
[13] T. Hu, J. Zhang, W. Kong, S. Yang and L. Cao, "Switch competing migration algorithm based on process optimization in SDN," Journal on Communications, vol. 08, pp. 213-222, 2017.
[14] J. Zhao, B. Zhang, L. Wang, H. Qu and L. Zheng, "Dynamic switch migration algorithm in software defined networks based on Q-learning," TV Engineering, vol. 06, pp. 68-72, 2016.
[15] A. G. Barto, "Reinforcement learning," A Bradford Book, vol. 15, no. 7, pp. 665-685, January 1998.
[16] R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction," in IEEE Transactions on Neural Networks, vol. 9, no. 5, pp. 1054-1054, Sept. 1998.
[17] S. Wang, H. Liu, P. H. Gomes and B. Krishnamachari, "Deep Reinforcement Learning for Dynamic Multichannel Access in Wireless Networks," IEEE Transactions on Cognitive Communications and Networking, vol. 4, no. 2, pp. 257-265, June 2018.
[18] [18] HAA. Al-Rawi, A. N. Ming and KLA. Yau, "Application of reinforcement learning to routing in distributed wireless networks: a review, " Artificial Intelligence Review, vol. 43, no.3, pp. 383-416, March 2013.
[19] Ryu SDN Framework, http://osrg.github.io/ryu/.
[20] Open Networking Foundation, https://www.opennetworking.org/.
[21] Mininet, http://mininet.org/.
[22] Open vSwitch, http://openvswitch.org/.
[23] iPref, https://iperf.fr/.